TARGETED PROPERTY-BASED TESTING

Andreas Löscher, Konstantinos Sagonas
andreas.loscher@it.uu.se, kostis@it.uu.se

Department of Information Technology
Uppsala University
Sweden
Outline

• Random Property-Based Testing
• Motivation
• Targeted Property-Based Testing

• Case Studies
• Concluding Remarks
Property-Based Testing

- High-level, semi-automatic, black-box testing technique.
- Testing user-specified properties of the SUT.
- Examples:
  - QuickCheck (Haskell)
  - ScalaCheck (Scala)
  - PropEr (Erlang)
  - ...

PropEr
A QuickCheck-Inspired Property-Based Testing Tool for Erlang
Random Property-Based Testing

- PBT tool provides:
  - Random generators for basic types
  - Language to write more complex generators

- PBT tool automatically tests these properties
  - Generate wide range of random inputs
  - Run the SUT with these inputs
  - Check if the properties hold
Random Property-Based Testing

**Generator**

\[ \text{prop_list_reverse}() \rightarrow \]

\[ \forall L, \text{proper_types: list(integer())}, \]

\[ \text{lists: reverse(lists: reverse(L)) == L}. \]

**Property should hold for all** \( L \)

**General Property**
Random Property-Based Testing

L = []
L = [2]
L = [-5,-1,-8,1]
L = [16,3,-23]
L = [38,29,-28,12,-11,-3,-28,-6,9,-16,4,4]
...

1> proper:quickcheck(example:prop_list_reverse(), 1000).
.................... 1000 dots ........................
OK: Passed 1000 test(s).
Graph Generator

\texttt{graph}(N) \rightarrow \\
\text{Vs} = \text{lists:seq}(1, N), \\
\text{?LET}(\text{Es, proper_types:list(edge(Vs))}, \\
\{\text{Vs, lists:usort(Es)}\}).

\texttt{edge}(\text{Vs}) \rightarrow \\
\text{?SUCHTHAT}\{\text{V1, V2}, \{\text{oneof(Vs), oneof(Vs)}\}, \\
\text{V1} < \text{V2}\)
Distance From Sink

On this graph, the maximum distance to sink is 4.

Is there a network with 42 nodes where the maximum distance to the sink > 21?
**Distance From Sink**

```prolog
prop_length() ->
  ?FORALL(G, graph(42),
    begin
      L = lists:max(distance_from_sink(G)),
      L < 21
    end).
```
Distance From Sink

1> proper:quickcheck(example:prop_length(), 100000).

................... 100000 dots ........................

OK: Passed 100000 test(s).

Same result for 1000 repetitions.

But we know that the property does not hold for some graphs.
Possible Solutions

- Write more involved generators?
- Guide input generation?
Possible Solutions

- Write more involved generators?
  - Guide input generation!
  - Using a search strategy.
Targeted Property-Based Testing

\texttt{prop\_length() ->}

\texttt{?FORALL(G, graph(42),}

\texttt{\hspace{1cm}begin}

\texttt{\hspace{2cm}L = lists:} \texttt{max(distance\_from\_sink(G))}

\texttt{\hspace{2cm}L < 21}

\texttt{\hspace{1cm}end).}

- Use a search strategy to find a \texttt{G} that falsifies the property.
- Observe the relationship between \texttt{G} and \texttt{L}. 
Targeted Property-Based Testing

- Combine Search Techniques with Property-Based Testing.
- Guide input generation towards input with high probability of failing.

- Gather information during test execution in form of utility values (UVs).
- UVs capture how close input came to falsifying a property.
prop_length_hc() ->
  ?FORALL(G, graph(42),
    begin
      L = lists:max(distance_from_sink(G)),
      L < 21
    end).

Targeted Property-Based Testing
Targeted Property-Based Testing

\texttt{prop\_length\_hc()} \rightarrow

\texttt{?FORALL(G, graph(42), begin Utility}

\texttt{\texttt{UV} = lists:max(distance\_from\_sink(G)), UV < 21 end).}
Targeted Property-Based Testing

prop_length_hc() ->
  ?FORALL(G, graph(42),
  begin
    Utility Values
    UV = lists:max(distance_from_sink(G)),
    ?MAXIMIZE(UV),
    UV < 21
  end).

Search Target Utility Values
Targeted Property-Based Testing

\[ \text{prop	extunderscore length	extunderscore hc}() \rightarrow \]
\[ \text{FORALL}(G, \text{TARGET(graph(42))), begin} \]
\[ \text{UV} = \text{lists:max}(\text{distance	extunderscore from	extunderscore sink}(G)), \]
\[ \text{MAXIMIZE(UV)}, \]
\[ \text{UV} < 21 \]
\[ \text{end}) . \]
Targeted Property-Based Testing

prop_length_hc() ->

?TARGET STRATEGY(hill climbing, ?FORALL(G, ?TARGET(graph(42))), begin
UV = lists:max(distance_from_sink(G)),
?MAXIMIZE(UV), UV < 21
end).

Search Strategy
Generator
the strategy controls
Utility Values
Search Target
Targeted Property-Based Testing

prop_length_hc() ->

?TARGET_STRATEGY(hill_climbing, ?FORALL(G, ?TARGET(graph_hc(42)))

begin

UV = lists:max(distance_from_sink(G)),

?MAXIMIZE(UV), UV < 21

end)

Now prop_length_hc fails after 17,666 tests (on average).
Targeted Property-Based Testing

- Hill Climbing requires a neighborhood function
  - which, currently, needs to be supplied by the programmer
  - remove and add some random edges from/to the graph

  Depends on the search strategy

- Hill Climbing can get stuck in local optima
  → Simulated Annealing is a better strategy
Targeted Property-Based Testing

\begin{align*}
\text{prop\_Target}() & \rightarrow \\
\text{?TARGET\_STRATEGY}(\text{SearchStrategy},) \\
\text{?FORALL}(\text{Input},) & \rightarrow \\
\text{?TARGET}(\text{Params}) \\
\text{begin} & \rightarrow \\
\text{UV} & = \text{SUT:run}(\text{Input}), \\
\text{?MAXIMIZE}(\text{UV}), & \rightarrow \\
\text{UV} & < \text{Threshold} \\
\text{end}() & .
\end{align*}
Case Study 1

Setup:
- Sensor network
- Random distribution of UDB server and client nodes
- Client node periodically sends messages to server node

Test:
- Has X-MAC for any network a duty-cycle > 25%?
  (duty-cycle ::= % time the radio is on)
Case Study 1

Random PBT
- Average amount of tests: 1188
- Average time per tests: 23.5s
- Mean Time to Failure: 7h46m

Targeted PBT
- Average amount of tests: 200
- Average time per tests: 40.6s
- Mean Time to Failure: 2h12m
Case Study 3

• Definitions for an abstract machine.
• Test: Do these definitions fulfill a certain security criteria? (Noninterference)

Case Study 3

Random PBT

- **Naive**: generate random programs
- **ByExec**: generate program step-by-step one instruction at a time; new instruction should not crash program

<table>
<thead>
<tr>
<th></th>
<th>Random PBT</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Naive</td>
<td>ByExec</td>
</tr>
<tr>
<td>ADD</td>
<td>2234,08ms</td>
<td>312.97ms</td>
</tr>
<tr>
<td>LOAD</td>
<td>324028,34ms</td>
<td>987,91ms</td>
</tr>
<tr>
<td>STORE A</td>
<td>timeout</td>
<td>4668,04ms</td>
</tr>
</tbody>
</table>
Targeted PBT

- **List**: programs are a list of instructions; using the built-in list generator for Simulated Annealing
- **ByExec**: neighbor of a program is a program with one more instruction

<table>
<thead>
<tr>
<th></th>
<th>Random PBT</th>
<th>Targeted PBT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Naive</td>
<td>ByExec</td>
</tr>
<tr>
<td>ADD</td>
<td>2234,08</td>
<td>312,97</td>
</tr>
<tr>
<td>LOAD</td>
<td>324028,34</td>
<td>987,91</td>
</tr>
<tr>
<td>STORE A</td>
<td>–</td>
<td>4668,04</td>
</tr>
</tbody>
</table>
Case Study 3

hand written; ca. 30 lines of additional code

<table>
<thead>
<tr>
<th></th>
<th>PBT</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Naive</td>
<td>ByExec</td>
</tr>
<tr>
<td>ADD</td>
<td>2234,08</td>
<td>312,97</td>
</tr>
<tr>
<td>LOAD</td>
<td>324028,34</td>
<td>987,91</td>
</tr>
<tr>
<td>STORE A</td>
<td>–</td>
<td>4668,04</td>
</tr>
</tbody>
</table>
Concluding Remarks

• Framework for Targeted Property-Based Testing.
• High-level expressive language for specifying properties.
• Compatible with random PBT.
• Two built-in strategies: hill climbing + simulated annealing.
• Infrastructure for additional search strategies.
• Fully integrated into PropEr.

PropEr
A QuickCheck-Inspired Property-Based Testing Tool for Erlang